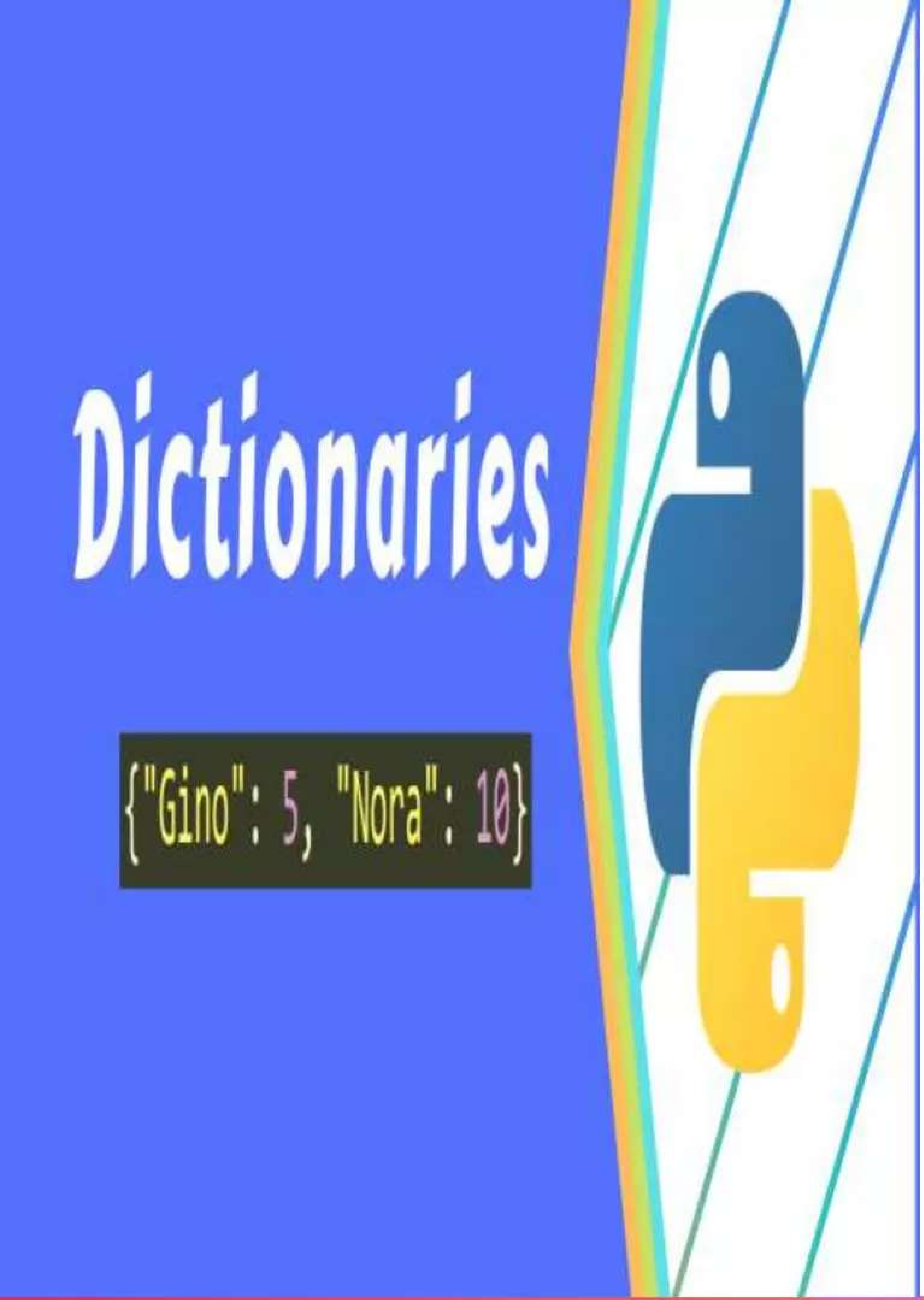
PYTHON DICTIONARY

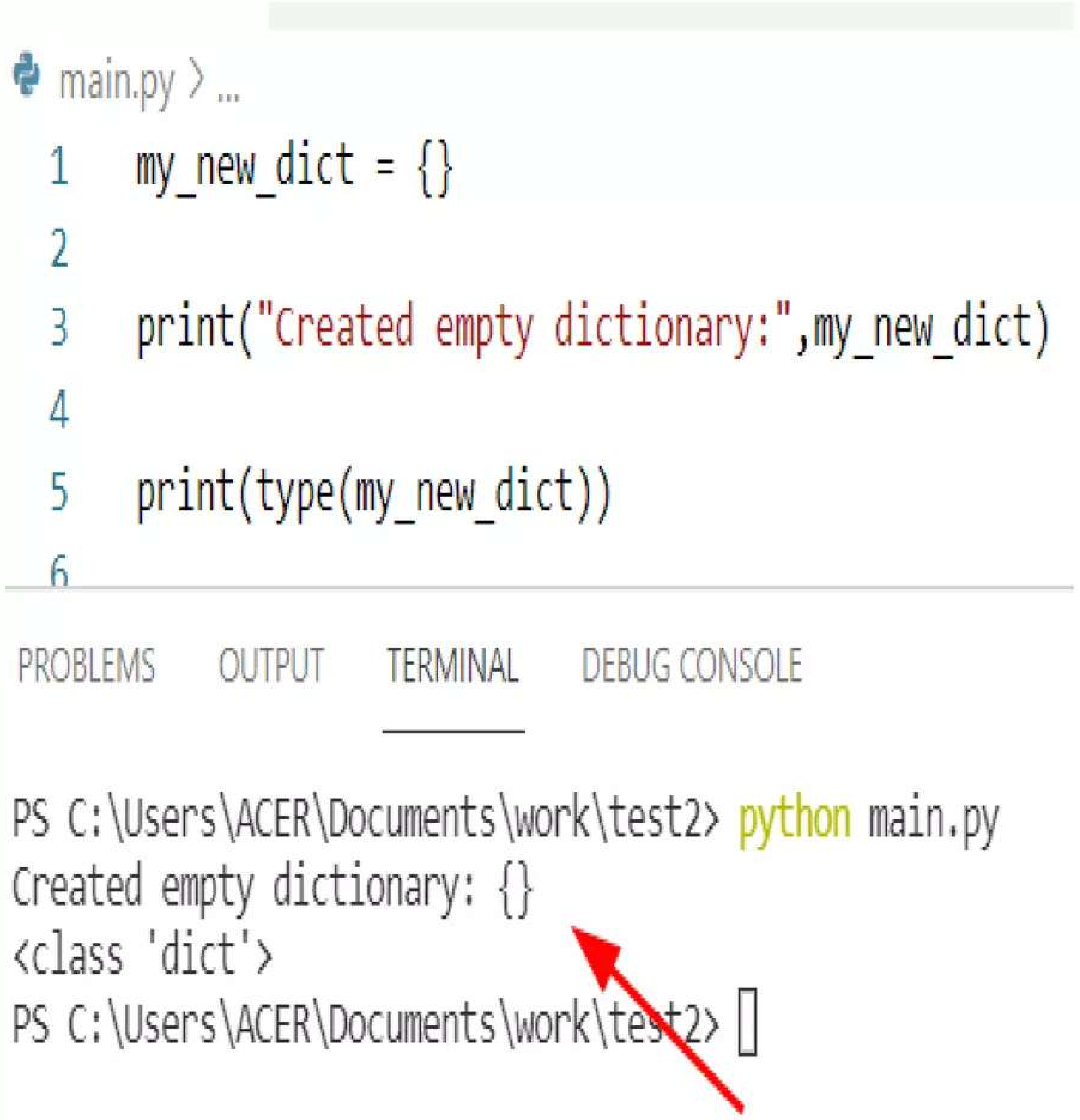


INTRODUCTION

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds key: value pair. Key-value is provided in the dictionary to make it more optimized.

CREATING A DICTIONARY

- Creating a dictionary is as simple as placing simple items inside curly braces{ } separated by comma.
- Each element in a dictionary is represented by a KEY:VALUE pair.
- While values can be of any data type and can repeat keys must be of immutable type and must be unique.



main.py > ...

```python
my_new_dict = {}

print("Created empty dictionary:",my_new_dict)

print(type(my_new_dict))
```

```
PS C:\Users\ACER\Documents\work\test2> python main.py
Created empty dictionary: {}
<class 'dict'>
PS C:\Users\ACER\Documents\work\test2> 
```

ELEMENTS IN DICTIONARY

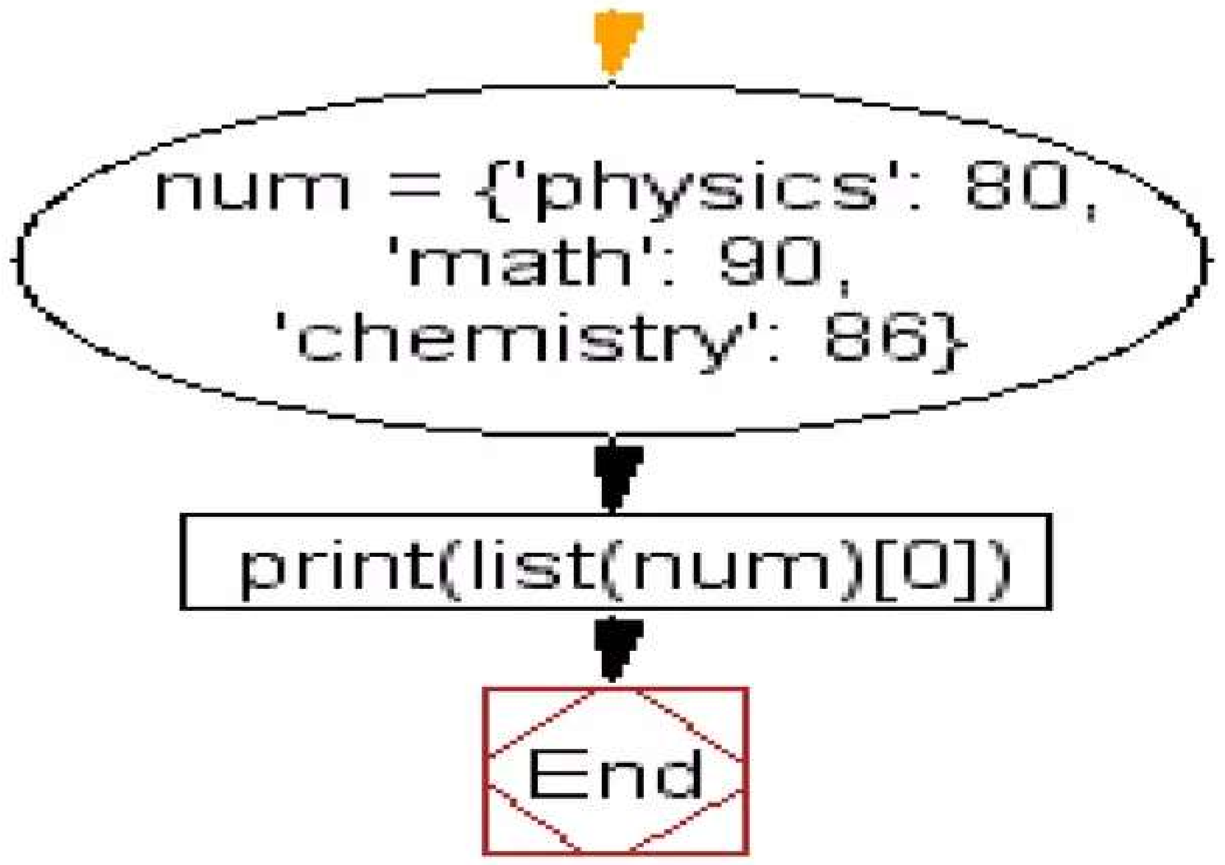
Accessing

Changing

Removing

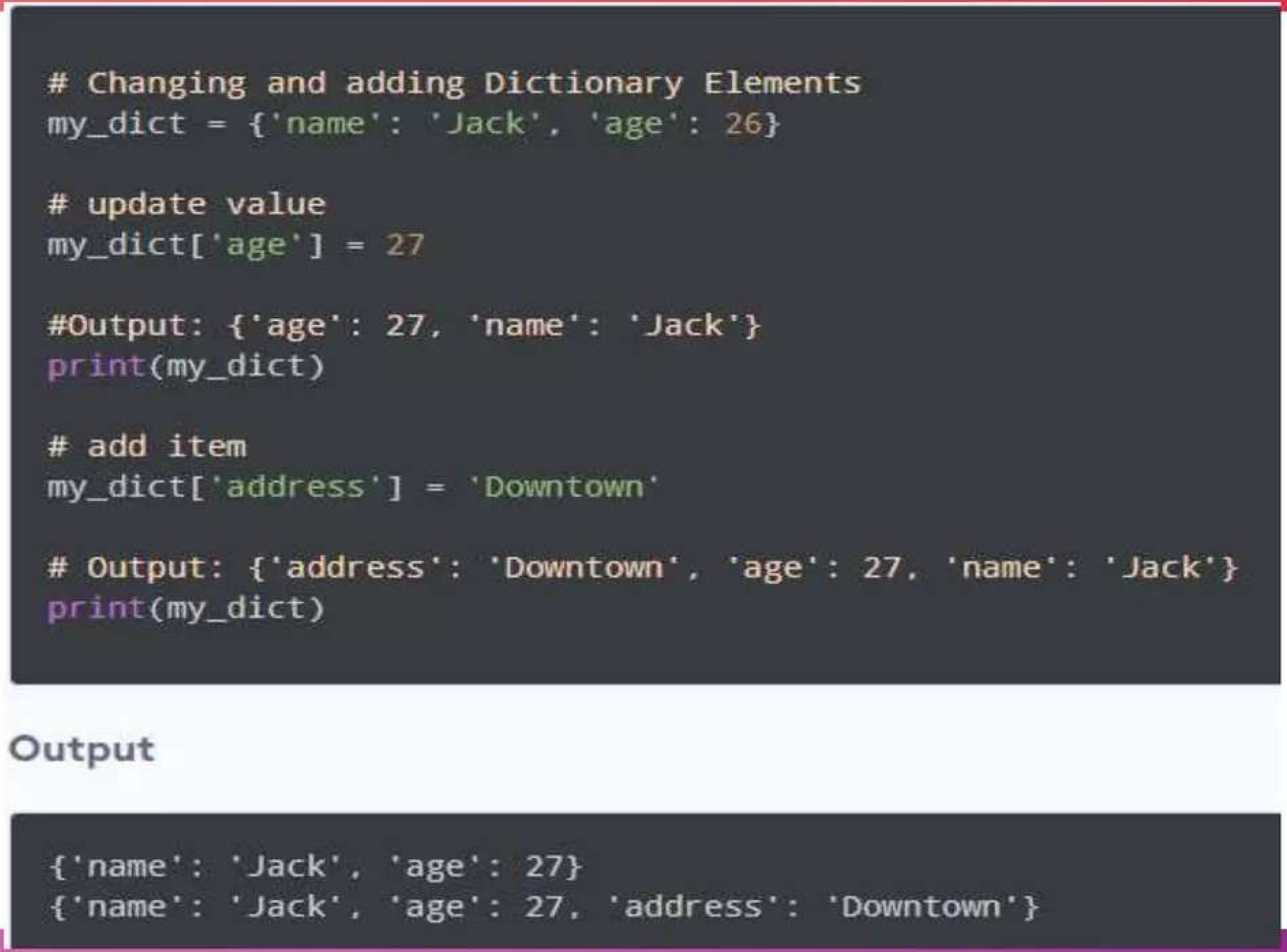
ACCESSING ELEMENTS FROM DICTIONARY

- While indexing is used other data types to access values, a dictionary uses keys.
- Keys can be used either inside square brackets [] or with the get() methods.
- If we use the square brackets [] , key error is raised in case a key is not found in the dictionary.



CHANGING AND ADDING DICTIONARY ELEMENTS

- Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.
- If the key is already present, then the existing value gets updated. In case the key is not present, a new (**key: value**) pair is added to the dictionary

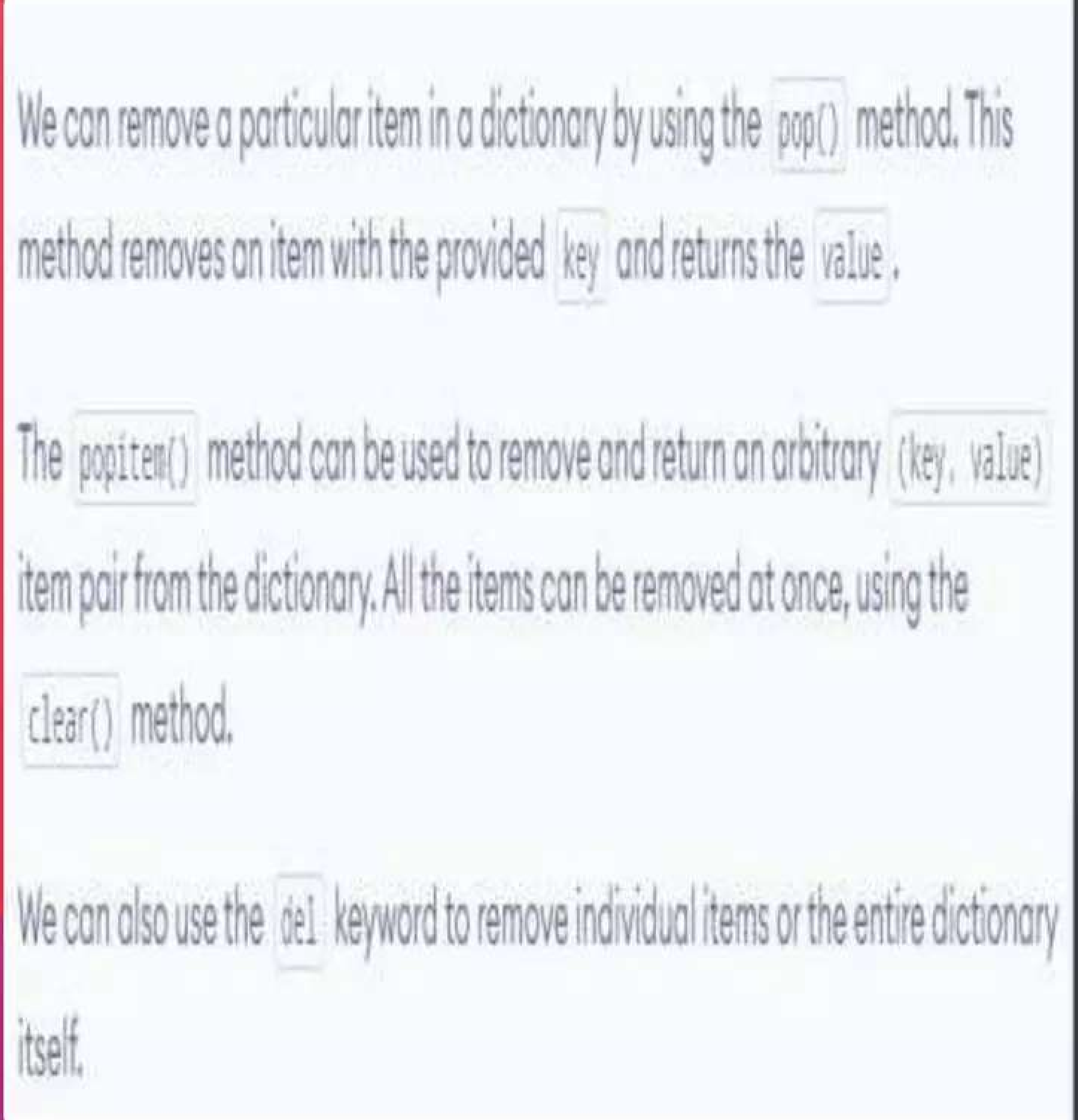


```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Output

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

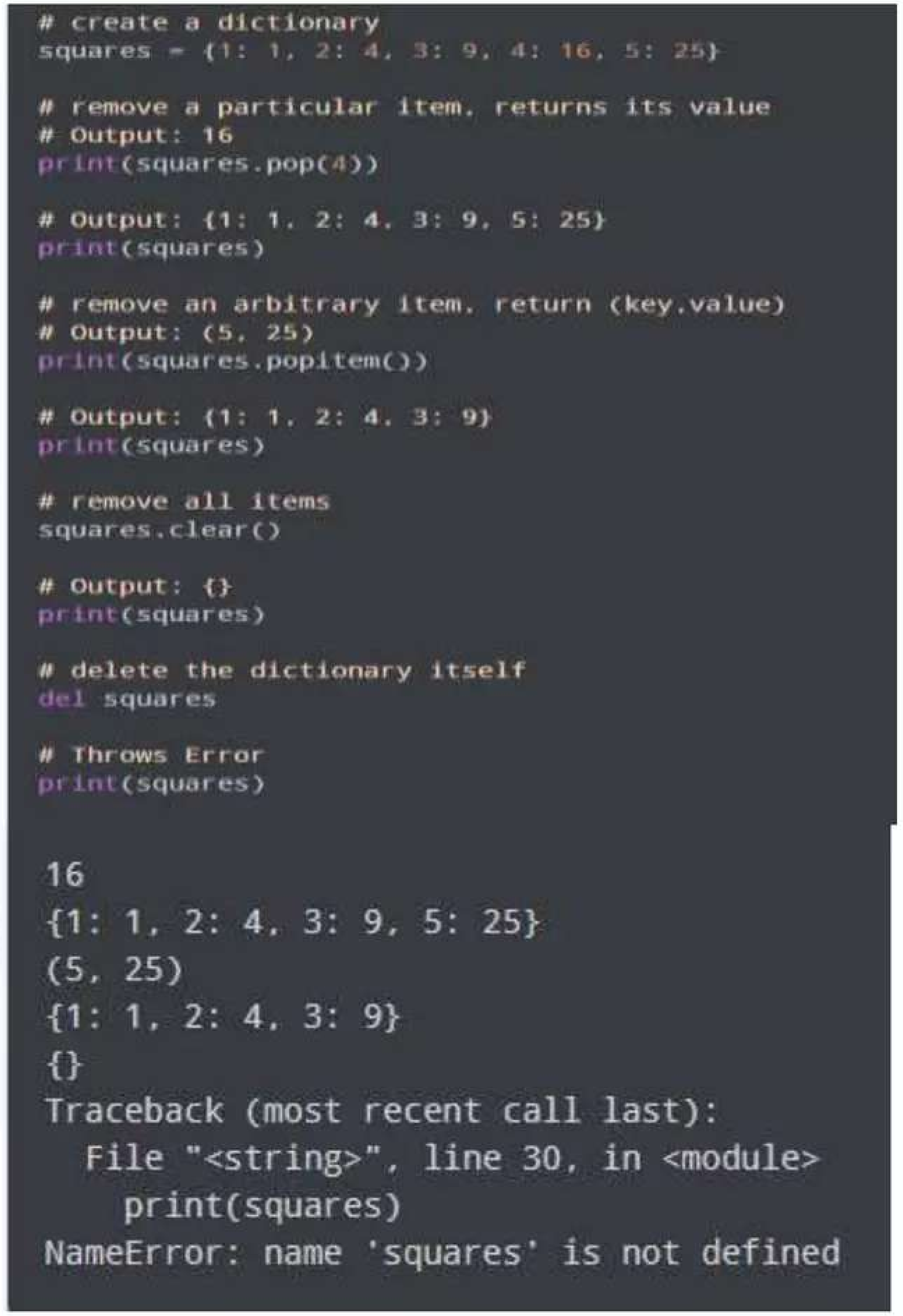
REMOVING ELEMENTS FROM DICTIONARY



We can remove a particular item in a dictionary by using the `pop()` method. This method removes an item with the provided `key` and returns the `value`.

The `popitem()` method can be used to remove and return an arbitrary `(key, value)` item pair from the dictionary. All the items can be removed at once, using the `clear()` method.

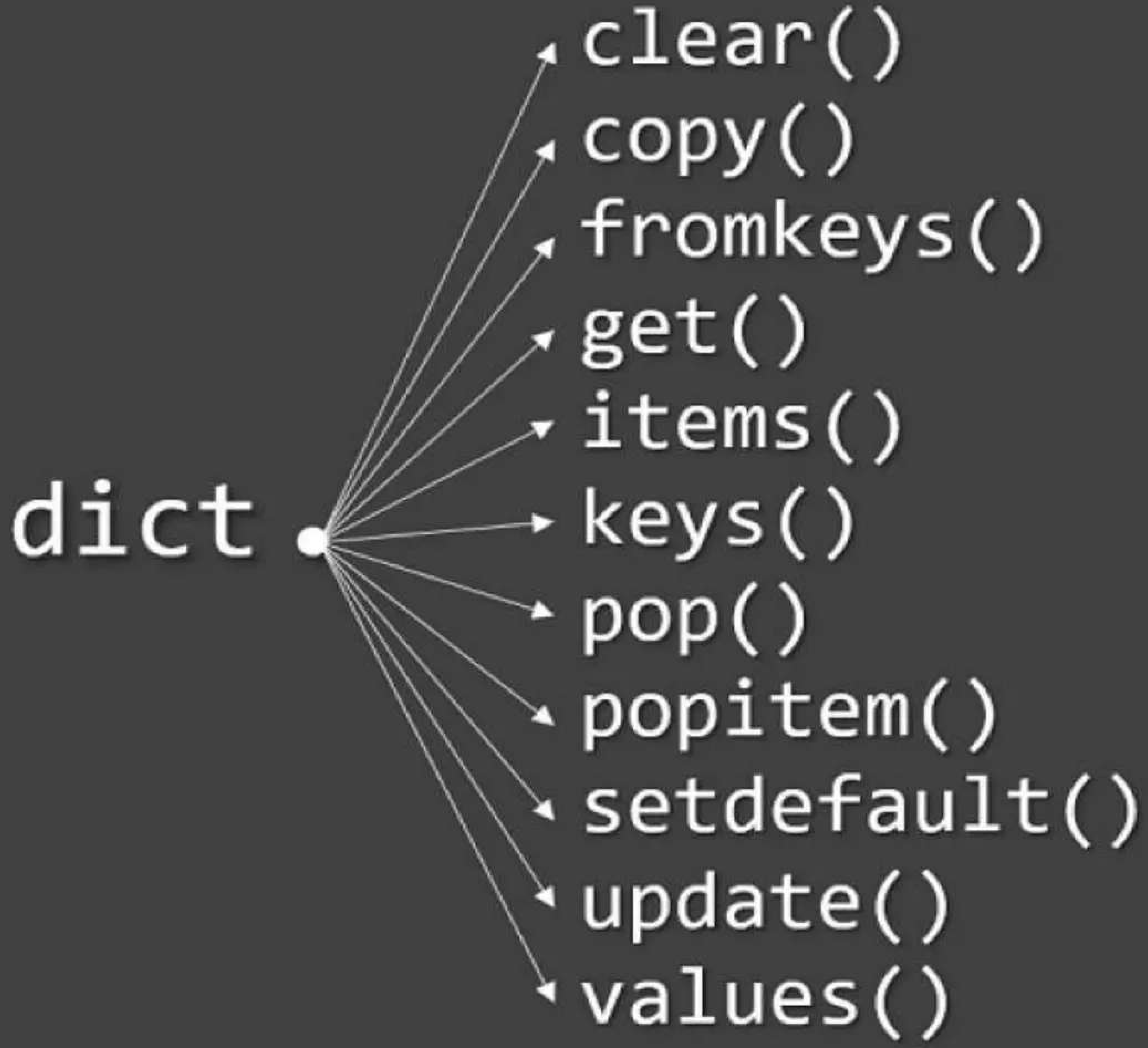
We can also use the `del` keyword to remove individual items or the entire dictionary itself.



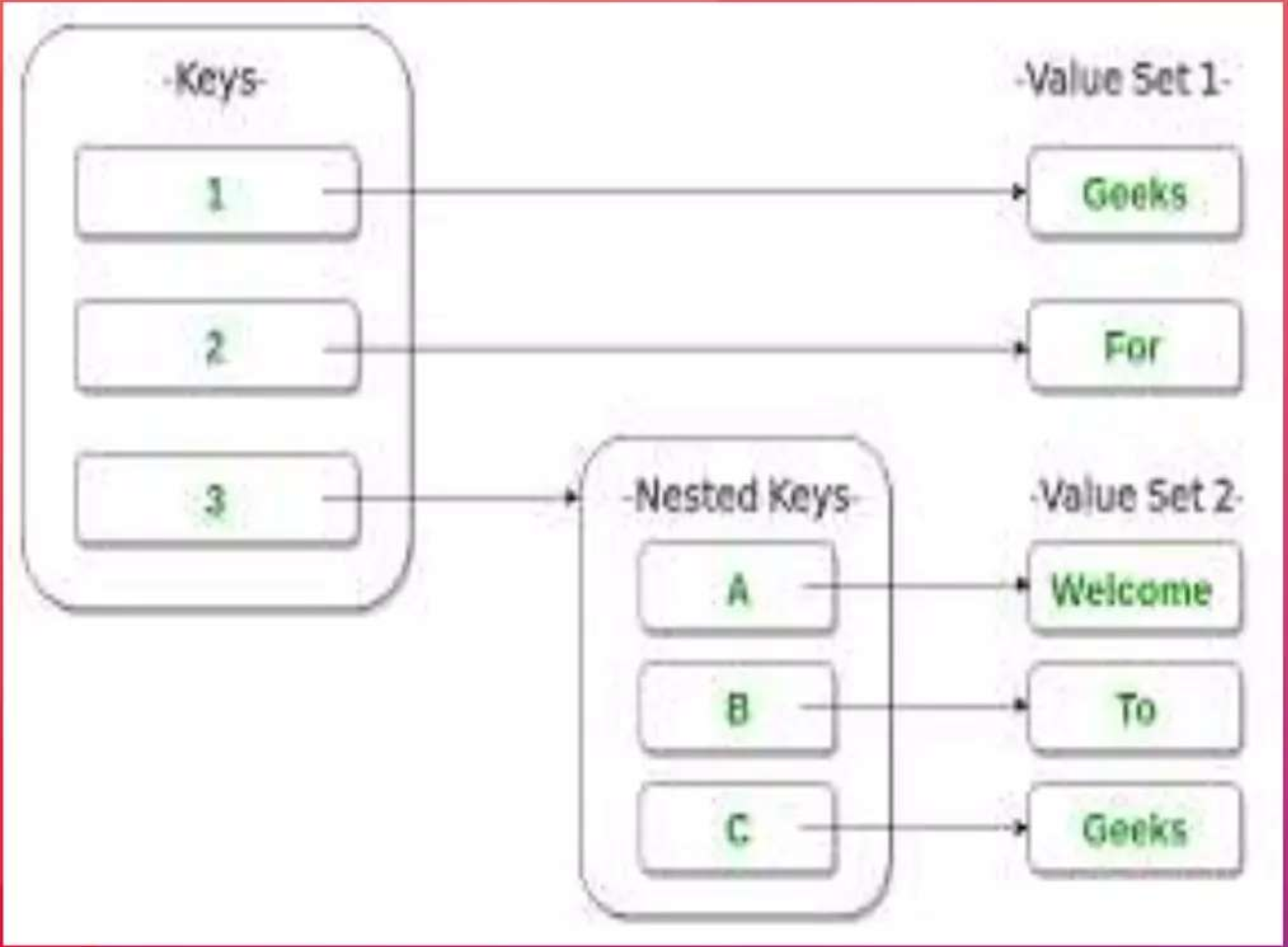
```python
# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(squares)

# remove all items
squares.clear()

# Output: {}
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)
```

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

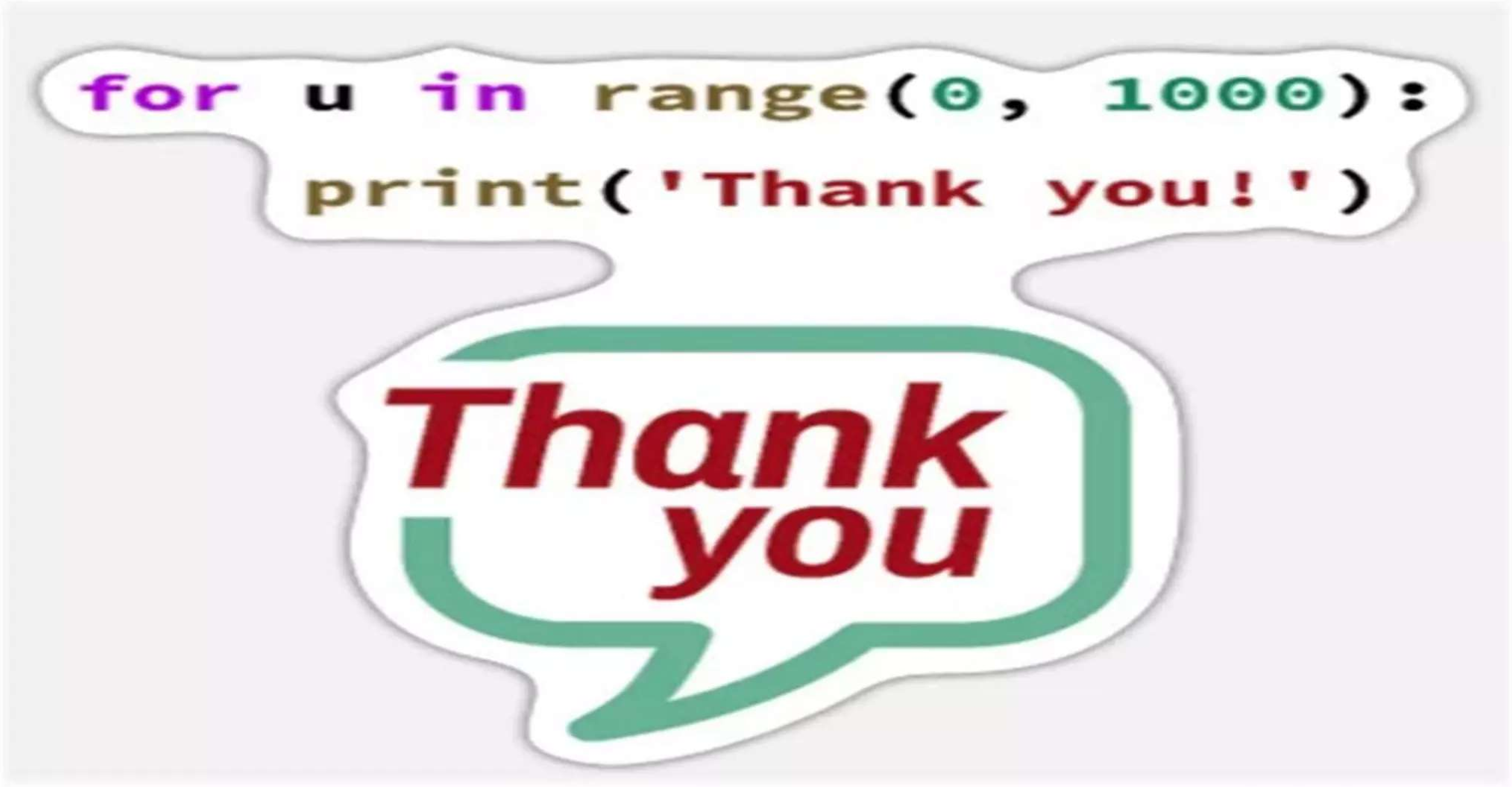
Python Dictionary Methods



NESTED DICTIONARY



Nesting Dictionary means putting a dictionary inside another dictionary. Nesting is of great use as the kind of information we can model in programs is expanded greatly.



```
for u in range(0, 1000):
    print('Thank you!')
```